Refine Search

Search Results -

Term	Documents
(17 AND 15).USPT.	11
(L15 AND L17).USPT.	11

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L18

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Tuesday, April 20, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L18	l15 and L17	11	L18
L17	L16.ab.	688	L17
L16	access$ near2 resource$1	7058	L16
L15	l9 and l14	585	L15
L14	call$	819035	L14
L13	l9 and l10	24	L13
L12	l9 and l10 and L11	24	L12
L11	process$	1957178	L11
L10	system call	6941	L10
L9	l7 and L8	856	L9
L8	counter	377522	L8
L7	l4 same L5	1448	L7
L6	l4 and L5	20848	L6

h e b b cg b e e ch

L5	(resource$1 or memor$) near3 access$	146225	L5
L4	number near1 time$1	104979	L4
L3	l1 and L2	1	L3
L2	generat$	1303163	L2
L1	6253225.pn.	1	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

Refine Search

Search Results -

Term	Documents
(17 AND 15).USPT.	11
(L15 AND L17).USPT.	11

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L18

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Tuesday, April 20, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L18	l15 and L17	11	L18
L17	L16.ab.	688	L17
L16	access$ near2 resource$1	7058	L16
L15	l9 and l14	585	L15
L14	call$	819035	L14
L13	l9 and l10	24	L13
L12	l9 and l10 and L11	24	L12
L11	process$	1957178	L11
L10	system call	6941	L10
L9	l7 and L8	856	L9
L8	counter	377522	L8
L7	l4 same L5	1448	L7
L6	l4 and L5	20848	L6

h e b b cg b e e ch

L5	(resource$1 or memor$) near3 access$	146225	L5
L4	number near1 time$1	104979	L4
L3	l1 and L2	1	L3
L2	generat$	1303163	L2
L1	6253225.pn.	1	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

Refine Search

Search Results -

Term	Documents
(9 AND 10).USPT.	27
(L9 AND L10).USPT.	27

Database:
- US Pre-Grant Publication Full-Text Database
- US Patents Full-Text Database
- US OCR Full-Text Database
- EPO Abstracts Database
- JPO Abstracts Database
- Derwent World Patents Index
- IBM Technical Disclosure Bulletins

Search: L11

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Tuesday, April 20, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L11	l9 and L10	27	L11
L10	compar$	1536757	L10
L9	l7 and L8	45	L9
L8	resource$1 near2 generat$	1579	L8
L7	l5 and L6	172	L7
L6	identifie$1	483717	L6
L5	l3 and l4	207	L5
L4	resource$1 near3 version$1	277	L4
L3	l1 or L2	146225	L3
L2	access$ near3 memor$	142219	L2
L1	access$ near3 resource$1	8648	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

First Hit Fwd Refs
End of Result Set

Generate Collection Print

L2: Entry 1 of 1 File: USPT Jun 26, 2001

US-PAT-NO: 6253225
DOCUMENT-IDENTIFIER: US 6253225 B1

TITLE: Process executing method and resource accessing method in computer system

DATE-ISSUED: June 26, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Nakahara; Masahiko	Yokohama			JP
Iwasaki; Masaaki	Tachikawa			JP
Takeuchi; Tadashi	Yokohama			JP
Nakano; Takahiro	Yokohama			JP
Serizawa; Kazuyoshi	Hadano			JP
Taguchi; Shihoko	Kawasaki			JP

US-CL-CURRENT: 718/100; 718/102, 718/104

CLAIMS:

What is claimed is:

1. A process executing method for executing a given one of a plurality of processes in a computer system by using one shared resource which is accessed by said plurality of processes executed on a processor, said method comprising the steps of:

a) disabling abortion of said given one process;

b) disabling preemption of said given one process;

c) processing said shared resource for use by said given one process after disabling preemption, and enabling preemption after processing said shared resource;

d) disabling preemption before succeeding processing said shared resource and enabling preemption after processing said shared resource;

e) enabling abortion of said given one process after enabling preemption; and

f) after enabling abortion, executing a forcive termination request issued for said given one process during a period in which said given one process was in an abort-disabled state.

2. A process executing method according to claim 1, further comprising the steps of:

registering processing requests for use of said shared resource issued by processes other than said given one process into a first queue; Q for process request stored.

retrieving one of the processing requests registered in said first queue by said given one process; get the request

after performing said steps a) to e), registering a completion message informing completion of said one of the processing requests into a second queue; and complete Queue

retrieving said completion message registered in said second queue by a process which issued said one of the processing requests. send a message to the process

First Hit Fwd Refs

Generate Collection Print

L22: Entry 11 of 20 File: USPT Sep 1, 1998

DOCUMENT-IDENTIFIER: US 5802590 A
TITLE: Method and system for providing secure access to computer resources

Abstract Text (1):
A method and system for allowing processes to access resources. A kernel of an operating system maintains a system-wide resource table. This resource table contains resource entries. When a resource is allocated, the kernel generates a key for the resource. The key is a very large number so as to prevent a malicious process from gaining unauthorized access to the resource. The kernel also hashes the key to generate an index into the resource table that is used as a handle. The kernel stores the key in a resource entry that is indexed by the handle. The handle.backslash.key pair is sent to a process. The process accesses the resources by passing handle.backslash.key pairs to the kernel. The kernel compares the passed key with a key that is stored in the resource entry referenced by the passed handle. When the stored key and the passed key match, the process is allowed to access the resource. When the stored key and the passed key do not match, the kernel rehashes the passed key to generate a new handle. The kernel then searches starting at the index of the new handle for a resource entry with a key that matches the passed key. When a key matches the passed key, the process is allowed to access the resource, and the index for the resource entry is returned to the process so that the process can use the index as a handle to access the resource on subsequent resource access requests. When the passed key does not match a key, the process is denied access to the resource.

Brief Summary Text (7):
To access a resource after it is allocated, a process passes the resource identifier (handle) to the kernel. The kernel uses the handle to access the entry in the resource table that corresponds to the resource. As a process requests the allocation of additional resources, the kernel adds resource table entries and returns the handles to the process.

Brief Summary Text (8):
The use of a handle as a resource identifier is advantageous because it imposes little processing overhead upon the system. Because the handle directly indexes the desired entry of the resource table, the kernel can quickly locate the requested resource. However, the use of a handle as a resource identifier has shortcomings. First, when each process has its own resource table, a handle cannot be simply passed from one process to another to share a resource. Rather, the server process notifies the kernel that one of its resources is to be shared with a client process. The kernel then adds a new entry to the resource table of the client process, sets the entry to identify the shared resource, and sends the handle to the client process. The server process can then access the shared resource using the handle into its resource table, and the client process can access the shared resource using the handle for the newly added entry into its resource table. Second, the use of handles can impose significant memory overhead upon the system. This memory overhead arises because the resource table size cannot be reduced as resources are deallocated. This overhead can be illustrated by assuming that a process has requested to deallocate a resource that corresponds to entry 120 of FIG. 1. To minimize memory usage, the kernel would remove entry 120 and compact the table accordingly (i.e., shift up those entries that are positioned beneath entry

120). However, if the resource table is compacted, then all handles that index entries that are moved would be incorrect. The results of such incorrect handles can be catastrophic (e.g., erasing critical data or causing system failure).

Detailed Description Text (9):
When the client process desires to access the shared resource, the client process passes the handle.backslash.key pair 315 to the kernel. The kernel invokes the Verify Access Routine 316 to determine whether the client process is authorized to access the resource identified by the handle.backslash.key pair. The routine makes this determination by comparing the passed key to the key stored at the resource entry indexed by the passed handle. When a stored key matches the passed key, the kernel allows the client process to access the resource. However, when the passed key does not match, the Verify Access Resource Routine rehashes the passed key to produce a new index. The routine uses linear probing to search for a resource entry that contains a key that matches the passed key. When such a resource entry is not found, the client process is denied access to the resource. On the other hand, when such a resource entry is found, the client process is allowed to access the resource, and the Verify Access Resource Routine returns to the client process the index of the resource entry as a newly-generated handle 319. By providing the client process with the newly-generated handle, the kernel can, for subsequent resource access requests, minimize the need to rehash the key. By avoiding unnecessary rehashing, the present invention minimizes the amount of processing overhead imposed upon the computer system.

Detailed Description Text (14):
When the kernel attempts to access the resource table using the original handle, it discovers that this handle is positioned beyond the boundaries of the resource table. In response, the kernel rehashes the passed key of FIG. 5C (i.e., 5DE . . . 68) to generate a new handle for the key. In this case, the kernel generates a handle of 4125. The kernel then compares the passed key (i.e., 5DE . . . 68) to the key that is referenced by the handle 4125. In this case, the passed key matches the key in the resource entry indexed by the handle 4125 (i.e., 5DE . . . 68). Since the keys match, the client is authorized to access the resource. The kernel then returns the new handle (i.e., 4125) to the client process. As explained above, by informing the client process of the new handle, the kernel ensures that on subsequent resource access requests from this client process, the kernel will first attempt to access the resource using the new handle. Upon receipt of this new handle, the client process replaces the old handle (i.e., 4298) with the new handle (i.e., 4125). The resulting handle.backslash.key pair is illustrated in FIG. 5D. By utilizing handle.backslash.key pairs in this manner, the present invention provides secure access to resources in a manner that reduces both processing and memory overhead.

Detailed Description Text (17):
FIG. 8 is a flow diagram of the computer-implemented steps of a client process in the preferred embodiment of the present invention. The client process receives a handle.backslash.pair key for a shared resource from a server process and uses this handle.backslash.key pair to access the shared resource. The client process first receives a handle.backslash.key pair from a server process (step 810). The client process attempts to access the resource by invoking an access resource routine of the kernel passing the handle.backslash.key pair (step 820). The kernel performs the necessary processing to give the client process access to the shared resource. The kernel returns a new handle for the key when the client process has attempted to access a resource using an old handle. If the kernel denies the client process access to the resource, because a resource identified by the key has not been allocated (e.g., attempted forging), the client process generates a resource denied error message (step 830). If the kernel returns a new handle, the client process replaces the old handle with the new handle (steps 840-850). By updating the handle in this manner, the client process ensures that the kernel will, on subsequent resource access requests, use a more currently generated handle. The client process

can then access the resource (step 860).

CLAIMS:

13. A method in a computer system for identifying a resource, the method comprising the steps of:

generating a handle.backslash.key pair, the key being generated as a non-forgeable identifier of the resource so that possessors of the generated key can use the key to indicate authority to access the resource, the handle being generated to identify a resource entry that identifies the resource, the resource entry containing the generated key; and

when accessing the resource,

receiving a handle.backslash.key pair;

when the resource entry identified by the received handle contains a key that matches the received key, indicating that the resource has been identified; and

when the resource entry identified by the received handle contains a key that does not match the received key,

generating a new handle for the passed key; and

when the resource entry identified by the new handle contains a key that matches the received key, indicating that the resource has been identified.

(1) Resource identifier and Generation

First Hit Fwd Refs

Generate Collection Print

L7: Entry 3 of 5 File: USPT Sep 1, 1998

DOCUMENT-IDENTIFIER: US 5802590 A
TITLE: Method and system for providing secure access to computer resources

Abstract Text (1):
A method and system for allowing processes to access resources. A kernel of an operating system maintains a system-wide resource table. This resource table contains resource entries. When a resource is allocated, the kernel generates a key for the resource. The key is a very large number so as to prevent a malicious process from gaining unauthorized access to the resource. The kernel also hashes the key to generate an index into the resource table that is used as a handle. The kernel stores the key in a resource entry that is indexed by the handle. The handle.backslash.key pair is sent to a process. The process accesses the resources by passing handle.backslash.key pairs to the kernel. The kernel compares the passed key with a key that is stored in the resource entry referenced by the passed handle. When the stored key and the passed key match, the process is allowed to access the resource. When the stored key and the passed key do not match, the kernel rehashes the passed key to generate a new handle. The kernel then searches starting at the index of the new handle for a resource entry with a key that matches the passed key. When a key matches the passed key, the process is allowed to access the resource, and the index for the resource entry is returned to the process so that the process can use the index as a handle to access the resource on subsequent resource access requests. When the passed key does not match a key, the process is denied access to the resource.

Brief Summary Text (5):
Processes frequently need to share resources. For example, a spreadsheet program may need to pass spreadsheet data to a word processing program so that the word processing program can incorporate the spreadsheet data into a document. The spreadsheet program can pass the data to the word processing program using shared memory. To pass the data, the spreadsheet program requests the kernel to allocate a block of memory for use as shared memory. In response, the kernel allocates the memory, maps the block of memory into the address space of the spreadsheet program, and returns to the spreadsheet program a resource identifier that identifies the block of memory. The spreadsheet program then writes the spreadsheet data to the block of memory and requests the kernel to allow the word processing program to share the block of memory. The kernel generates another resource identifier that the word processing program can use to access the block of memory. Once the block of memory is mapped into its address space, the word processing program can access the spreadsheet data.

Brief Summary Text (6):
To help manage the various resources, the kernel maintains a resource table for each process. For each allocated resource, the resource table contains an entry, which identifies the resource. FIG. 1 shows a conventional resource table 100 in which resources are identified by handles. Entry 120 corresponds to an allocated block of memory 130 and contains a pointer to the memory block. The resource identifier that the kernel returns to the process which requested allocation of the resource is an index into the resource table for the process. This index is also known as a handle. The handle 110 contains an index into the resource table. A process that requests the kernel to allocate a resource so that the process can

h e b b g e e e f c e e e ge

share the resource with other processes is known as a "server" process. Analogously, a process that utilizes a resource that is shared by a server process is known as a "client" process.

Brief Summary Text (7):
To access a resource after it is allocated, a process passes the resource identifier (handle) to the kernel. The kernel uses the handle to access the entry in the resource table that corresponds to the resource. As a process requests the allocation of additional resources, the kernel adds resource table entries and returns the handles to the process.

Brief Summary Text (8):
The use of a handle as a resource identifier is advantageous because it imposes little processing overhead upon the system. Because the handle directly indexes the desired entry of the resource table, the kernel can quickly locate the requested resource. However, the use of a handle as a resource identifier has shortcomings. First, when each process has its own resource table, a handle cannot be simply passed from one process to another to share a resource. Rather, the server process notifies the kernel that one of its resources is to be shared with a client process. The kernel then adds a new entry to the resource table of the client process, sets the entry to identify the shared resource, and sends the handle to the client process. The server process can then access the shared resource using the handle into its resource table, and the client process can access the shared resource using the handle for the newly added entry into its resource table. Second, the use of handles can impose significant memory overhead upon the system. This memory overhead arises because the resource table size cannot be reduced as resources are deallocated. This overhead can be illustrated by assuming that a process has requested to deallocate a resource that corresponds to entry 120 of FIG. 1. To minimize memory usage, the kernel would remove entry 120 and compact the table accordingly (i.e., shift up those entries that are positioned beneath entry 120). However, if the resource table is compacted, then all handles that index entries that are moved would be incorrect. The results of such incorrect handles can be catastrophic (e.g., erasing critical data or causing system failure).

Detailed Description Text (2):
The present invention is directed towards a method and system in a kernel of an operating system for providing secure access to computer system resources while minimizing overhead. In a preferred embodiment, the kernel maintains a system-wide resource table that is a hash table and that contains a resource entry corresponding to each resource allocated by the kernel. The allocated resources are identified by a kernel-generated resource identifier. The system of the present invention uses resource identifiers that contain both a handle and a key (a handle.backslash.key pair). The key is a very large number (e.g., 128 bits) that uniquely identifies the resource, and the handle contains an index into the resource table. The key is a randomly generated number that is sufficiently sized so as to render the forging of a key unlikely. The handle is generated by inputting the key into a hash function that generates an index into the resource table. Using linear probing hashing techniques, the kernel determines the index of the next available resource entry. A resource entry corresponding to the allocated resource is stored in the resource table at the determined index. The determined index is used as the handle. The stored resource entry contains a copy of the key. The resource identifier comprising the handle.backslash.key pair is passed to the process that requested the allocation of the resource.

Detailed Description Text (9):
When the client process desires to access the shared resource, the client process passes the handle.backslash.key pair 315 to the kernel. The kernel invokes the Verify Access Routine 316 to determine whether the client process is authorized to access the resource identified by the handle.backslash.key pair. The routine makes this determination by comparing the passed key to the key stored at the resource

entry indexed by the passed handle. When a stored key matches the passed key, the kernel allows the client process to access the resource. However, when the passed key does not match, the Verify Access Resource Routine rehashes the passed key to produce a new index. The routine uses linear probing to search for a resource entry that contains a key that matches the passed key. When such a resource entry is not found, the client process is denied access to the resource. On the other hand, when such a resource entry is found, the client process is allowed to access the resource, and the Verify Access Resource Routine returns to the client process the index of the resource entry as a newly-generated handle 319. By providing the client process with the newly-generated handle, the kernel can, for subsequent resource access requests, minimize the need to rehash the key. By avoiding unnecessary rehashing, the present invention minimizes the amount of processing overhead imposed upon the computer system.

Detailed Description Text (16):
FIG. 7 is a flow diagram of the computer-implemented steps for the Allocate Resource Routine of the kernel. The Allocate Resource Routine receives a resource specification as input and returns a resource identifier comprising a handle.backslash.key pair that corresponds to the resource. This handle.backslash.key pair references a resource entry corresponding to the resource. The routine begins by receiving the resource specification from a calling process (e.g., a server process) (step 710). The routine generates a key for the resource identifier and creates a handle for the key (steps 720 and 730). The key is preferably generated by a random number generator, and the handle is generated by a hash function for the resource table. The routine then stores the key in the resource entry of the resource table indexed by the handle (step 740). The routine then allocates the specified resource using standard and well-known techniques that are appropriate to the type of resource (step 750). Finally, the routine returns the handle.backslash.key pair to the calling process (step 750).

Detailed Description Text (17):
FIG. 8 is a flow diagram of the computer-implemented steps of a client process in the preferred embodiment of the present invention. The client process receives a handle.backslash.pair key for a shared resource from a server process and uses this handle.backslash.key pair to access the shared resource. The client process first receives a handle.backslash.key pair from a server process (step 810). The client process attempts to access the resource by invoking an access resource routine of the kernel passing the handle.backslash.key pair (step 820). The kernel performs the necessary processing to give the client process access to the shared resource. The kernel returns a new handle for the key when the client process has attempted to access a resource using an old handle. If the kernel denies the client process access to the resource, because a resource identified by the key has not been allocated (e.g., attempted forging), the client process generates a resource denied error message (step 830). If the kernel returns a new handle, the client process replaces the old handle with the new handle (steps 840-850). By updating the handle in this manner, the client process ensures that the kernel will, on subsequent resource access requests, use a more currently generated handle. The client process can then access the resource (step 860).

L18: Entry 15 of 22 File: USPT Nov 21, 1995

DOCUMENT-IDENTIFIER: US 5469556 A
TITLE: Resource access security system for controlling access to resources of a data processing system

Abstract Text (1):
A resource access security system for use in a data processing system for controlling access to resources correspondingly assigned to addresses in an address space of the data processing system by the use of descriptors. The descriptors correspondingly identify the resources and access to the resources is controlled by requiring the input of a descriptor of the resource to which access is sought. The resource access security system controls access to the resources by translating each descriptor being taught to gain access to a resource by use of a plurality of tables having stored therein user/job information, domain information and page information and a descriptor translator which controls the descriptor translation process. The descriptors are virtual addresses of addresses assigned to the resources of the data processing system.

Brief Summary Text (12):
The present invention provides a resource access security system for controlling access to resources including hardware, software, input/output ports and data in the form of databases of a data processing system by the use of descriptors which identify the resources.

Brief Summary Text (17):
To implement the orthogonal protection mechanisms of the present invention, all resources in the system including software, data in the form of databases, input/output ports connected to terminals, printers, external memories (disk drives), and hardware such as cache memory, internal memory or the like, are assigned a privilege level in addition to a classification level by use of descriptors. These descriptors each identify a resource and include information related to the privilege level, classification level, and address of the resource in the address space of the data processing system.

Brief Summary Text (18):
In the present invention, all resources needed to perform specific functions are organized into a domain which corresponds to a privilege level. Information identifying the domain to which the resource belongs is included in the descriptor. Also, as described above, each resource is assigned a classification level by organizing the resources into pages which correspond to the classification levels. Information identifying the classification level and page to which the resource belongs is included in the descriptor. Classification levels are not only assigned to software resources such as processes and databases, but also to input/output ports, and hardware such as terminals, printers, disk drives, etc.

Brief Summary Text (19):
The descriptor also includes information identifying an address in the address space to which the resource is assigned.

Brief Summary Text (20):
Essentially, the descriptor described above, including information identifying the

domain, page and address of the resource in the address space of the data processing system, forms a virtual address of the resource. Thus, in the present invention, when a descriptor is used to access a particular resource, the descriptor must be translated by a descriptor translator to obtain the real address assigned to the resource in the address space of the data processing system.

Brief Summary Text (21):
Therefore, the present invention provides a resource access security system to be used in a data processing system for controlling access to resources correspondingly assigned to addresses in an address space of the data processing system by the use of descriptors. The descriptors correspondingly identify the resources. Access to the resources is controlled by requiring the input of a descriptor corresponding to the resource to which access is sought and translating the descriptor so that only the address assigned to the resource in the address space of the data processing system is accessed.

Brief Summary Text (22):
The resource access security system of the present invention performs the translation of a descriptor by use of a user/job information table, including a plurality of user/job entries, each having stored therein a user/job pointer which points to a base address of a domain table to which the user/job has the privilege of access, a plurality of domain tables, each including a plurality of domain entries, wherein each domain entry has stored therein a domain pointer which points to a base address of a page table and a plurality of page tables, each having a plurality of pages wherein each page has stored therein a base address of a resource in the address space of the data processing computer to which access is sought.

Brief Summary Text (24):
Also included in the present invention is decoding means for decoding an instruction to be executed by the data processing system to determine whether the instruction is an instruction for accessing a resource. If the instruction is an instruction for accessing a resource, the decoding means obtains from external memory, or a register internal to the data processing system, an operand associated with the instruction, wherein the operand is a virtual address or descriptor identifying the resource.

Brief Summary Text (25):
A descriptor translating means is provided in the present invention for translating the descriptor obtained by the decoding means by using the user/job table, domain table and page table to thereby determine an address within the address space of the data processing system assigned to the resource to which access is sought. Thus, by use of the present invention, access is only permitted to the resource identified by the user/job, domain and page information.

Detailed Description Text (2):
FIG. 1 illustrates the resource access security system and the data processing system of the present invention. The total system shown in FIG. 1 provides a high degree of security by establishing as much process containment as is feasibly possible. Particularly, the present invention provides a separation of processor privilege levels from data classification levels by the use of descriptors which identify the resources of the data processing system.

Detailed Description Text (5):
Specifically, as shown in FIG. 1, the present invention includes an unsecured processor 10 and a secure processor 12. The unsecured processor 10 executes all unsecure instructions, and the secure processor 12 executes all secure instructions. The secure instructions include privileged instructions which access a resource to which access is controlled by the resource access security system of the present invention. Unsecured instructions are normal processing instructions

for processing data internal to the data processing system. The secure processor 12 is connected to memory management registers 14, secure register file 16, and shadow registers 18. A bus interface monitor 20 is provided for checking the validity of every access to resources (not shown) of the data processing system. The bus interface monitor 20 is connected to bus interface unit (BIU) 22, unsecured BIU registers 24, secure BIU registers 26, program counter 28, stack pointers 30, queue controller 32 which is connected to a queue 34, unsecured processor 10, and shadow registers 18. BIU 22 is also connected to the resources of the data processing system.

Detailed Description Text (6):
The memory management registers 14 are used to hold descriptor information of resources to which the data processing system presently has access. The secure register file 16 includes general purpose registers used by the secure processor 12 for storing and processing data. Shadow registers 18 holds the results of virtual to real address translations, classification tags, and access information of the resources of the data processing system of the present invention. The stack registers 30 hold a virtual address pointing to data of a domain (privilege level) page (classification) and offset of a resource in the memory of the data processing system of the present invention. The domain, page, and offset will be described in more detail below. Program counter 28 holds the virtual address of the next instruction to be executed.

Detailed Description Text (9):
As described above, queue controller 32 operates to perform a prefetch of instructions and associated operands through bus interface monitor 20. The prefetched instructions are stored in the queue 28. The instruction group decoder 36 fetches an instruction from the queue 34 and decodes the instruction to determine whether the instruction is a secure instruction or an unsecured instruction. Depending upon the type of instruction, operands are fetched either from the queue 34, the register file 40, the secure register file 16, or an external resource. If one of the operands is fetched from an external resource, its virtual address is fetched from either the queue 34, the register Files 40, 16, or the stack pointers 30. As indicated above, if the instruction is a secure instruction, the instruction is transferred to the secure processor 12, and if the instruction is an unsecured instruction, the instruction is transferred to the unsecured processor 10. The unsecured processor 10 operates in response to instructions transferred from the instruction group decoder 36. The instructions transferred from the instruction group decoder 34 are decoded by the unsecured processor 10 to determine the particular operation sought by the instruction. Thereafter, the unsecured processor 10 operates in response to the instruction transferred from the instruction group decoder 36 to perform the indicated operation. The secure processor 12 operates in response to instructions transferred from the instruction group decoder 36. The instructions transferred from the instruction group decoder 36 are decoded by the secure processor 12 to determine the operation indicated thereby and whether access to a resource is being sought. Thereafter, the secure processor 12 performs the operation indicated by the instruction and any descriptor translations so as to control access to the resources of the data processing system.

Detailed Description Text (14):
In the present invention, when data is transferred to the register file 40 from an external resource, a classification tag is used by the tag monitor 38 to maintain classification level identification of the data. Conversely, when data is transferred from the register file 40 to an external resource, the classification tag is used by the bus interface monitor 20 to ensure that the data is not written to a resource with a lower classification level. Particularly, data from the tag monitor 38 and the register file 40 is first transferred to the unsecured BIU registers 24, and then to memory, and vice-versa. The secure BIU registers 26 being connected to the bus interface monitor 20 and the bus interface unit 22 inputs and

h e b b g e e e f c e e e g

receives data to and from the secure processor 12, secure register file 16, the stack pointers 30, the queue controller 32, and the program counter 28. The bus interface unit 22 inputs and outputs data to and from the resources assigned to addresses in the address space of the data processing system of the present invention.

Detailed Description Text (15):
Control of access to the resources of the data processing system is controlled by the bus interface monitor 20. The bus interface monitor 20 responds to signals from the queue controller 32, the unsecured processor 10 and the secure processor 12. Depending upon the source of the control signal, the bus interface monitor 20 fetches a virtual address from either the queue controller 32, the unsecured processor 10 or the secure processor 12, or one of the stack pointers 30. The virtual address identifies the resource being accessed. The bus interface monitor 20 applies the virtual address to entries in the shadow registers 18 to determine whether information related to the virtual address, from a previous translation is stored therein. If information related to the virtual address is found in the shadow registers 18, then the bus interface monitor 20 also checks the shadow registers 18 for access permission information. Thus, if access is permitted, then the real address stored in the shadow register 18 related to the virtual address is transferred to the bus interface unit 22 and a bus cycle is initiated. The shadow registers 18 as indicated above are used to store data of previous translations of virtual addresses in order to speed the translation process by eliminating the need to perform another translation process when the virtual address of concern has been previously translated.

Detailed Description Text (16):
As described above, the tag monitor 38 is used to maintain the classification tags of data in the register file 40 to insure they are updated properly after each operation. Also, if data in the register file 32 is moved, its new location is tagged with the same classification as before by the tag monitor 38. The secure processor 12 shown in greater detail in FIG. 2 includes a secure operation decoder 50, secure microcode 52, secure microcode address generator and sequencer 54, and secure arithmetic logic unit (ALU) 56. The secure microcode 52 includes a descriptor translator module 52a for translating a descriptor (virtual address) of a resource to a real address.

Detailed Description Text (17):
Instructions from the instruction group decoder 36 are sequentially input to the secure operation decoder 50 and decoded thereby. Control signals are then supplied from the secure operation decoder 50 to the secure address generator and sequencer 54 which supplies sequentially generated addresses to the secure microcode 52 related to the operation indicated by the control signals in order to obtain the appropriate microcode subroutine. If a secure instruction which seeks access of a resource is decoded, then the descriptor translator module 52a is addressed by the secure address generator and sequencer. Control signals from the secure microcode 52 are applied to the secure arithmetic logic unit 56, secure register file 16, secure information register file 17 stack pointers 30, program counter 28, secure BIU registers 26, memory management registers 14, and shadow registers 18 in order to control the operations performed thereby.

Detailed Description Text (18):
The secure processor 12 makes use of the secure register file 16 for general storing processing operation. As described above, the secure register file 16 includes a plurality of registers. The secure ALU 56 performs functions on any two of the registers of the secure register file 16 such as add, subtract, compare, OR, XOR, and AND. Any of the registers of the secure register file 16 can also be rotated one bit to the left or right by the secure ALU 56. In addition to the secure register file 16, the secure processor 12 also has access to a plurality of memory management registers 14 which contain the system and global root pointers

and limit values, the virtual address of the current domains being accessed, their descriptor table root pointers and limit data, and the identification and classification and clearance level of the current user/job. The system and global root pointers and limit values are described below.

Detailed Description Text (19):
As described above, the present invention implements the resource access security control by implementing a virtual addressing system whereby each resource of the data processing system is identified by a descriptor which is a virtual address thereof. In order to access a particular resource by use of the present invention, the descriptor must be input by the user or job in association with an instruction and translated to obtain the real address of the resource. A descriptor as shown in FIG. 3 includes offset information identifying a user/job 60, offset information identifying a domain 62, offset information identifying a page 64, and offset information identifying the offset of an address 66. User/global information 68 is also included in the descriptor for identifying when a user or global access is being sought. A global access being an access not constrained by the particular user or job seeking access.

Detailed Description Text (21):
The resources are also organized into pages which correspond to the clearance levels or classifications of the data processing system, which may be, for example, unclassified, classified, secret and top secret. Thus, the page offset information 64 provides information identifying the page to which the resource belongs. The address offset information 66 provides an offset to an address in the address space assigned to the resource to which access is requested. The descriptor shown in FIG. 3 essentially is a virtual address of the resource in the address space of the data processing system. Access to the resources is controlled by assigning each of the resources to a descriptor. In order to gain access to a resource, the appropriate descriptor must be applied. The descriptor being applied is translated by the descriptor translator module 52a in order to obtain the real address of the resource.

Detailed Description Text (24):
In the present invention, a system root pointer is stored in a system root pointer register 75 in order to point to the base address of the user/job table 70. The maximum allowable offset into the user/job table is located within system limit register 75a. The user/job table 70 includes a plurality of user/job entries with each entry having stored therein a user/job pointer which points to a base address of a domain table, as well as a limit value which specifies the maximum allowable offset into the table. Each of the domain tables includes a plurality of domain entries each having stored therein a domain pointer which points to a base address of a page table, as well as a limit value which specifies the maximum allowable offset into the table. Each of the page tables includes a plurality of pages wherein each page has stored therein a base address of a resource of the data processing system.

Detailed Description Text (25):
FIG. 5 shows a flow chart of the descriptor translation process performed by the descriptor translator module 52a using the tables shown in FIG. 3. Initially, the bus interface monitor 22 detects a request to access a resource not represented by entries in the shadow registers 18. The virtual address of the access request is pushed on the stack by use of the stack pointers 30 and the request is then referred to the secure processor which obtains the descriptor thereof by use of stack pointers 30 (step 80).

Detailed Description Text (26):
Next, if the user/job offset information 60 of the descriptor is not greater than the system limit 75A, then it is applied to the user/job table pointed to by the system root pointer (step 82). Then a base address and limit of a domain table is

obtained from the user/job table 70 (step 84). If the domain offset information 62 of the descriptor is not greater than the limit value for the domain, it is applied to the domain table pointed to by the base address in step 84 (step 86). A base address and limit of a page table is then obtained (step 88). Thereafter, if the page offset information 64 is not greater than the limit value, it is applied to the page table identified by the base address of the page table obtained in step 88 (step 90) in order to obtain a base address of the resource to which access is sought (step 92). The base address of the requested resource obtained in step 92 is added to the address offset information 66 (step 94), in order to obtain a real address of the resource to which access is sought (step 96). In steps 82, 84 and 86, if any offset is greater than the corresponding limit value, the descriptor translation process is terminated (steps 83, 87 and 91).

Detailed Description Text (28):
Another embodiment of the present invention is shown in FIG. 6. In the tables shown in FIG. 6, the current user/job offset information of the current user/job of the data processing system is prestored in the user/job identification register of the memory management registers 14. This accessing technique is called a global access as described above. This arrangement eliminates the use of the user/job table 70 and replaces it with a user/job access list 100. The process for translating a descriptor by use of the tables shown in FIG. 6, is shown by a flow chart in FIG. 7. To obtain access to a particular resource of the data processing system by use of the tables shown in FIG. 6, the system need only apply the domain offset information 62, the page offset information 64, and the offset information 66 to the tables. A base address of a domain table 104 is pointed to by a domain table pointer stored in a global root pointer register 110 shown in FIG. 6. The maximum allowable offset into the global domain table is located in the global limit register 110A.

Detailed Description Text (29):
If domain offset address information 62 is not greater than the global limit 110A, it is applied to the domain table 104 indicated by the domain table pointer (step 200). A base address and limit of a page table 112 and information indicating a particular user/job access list 100 is obtained from the domain table (step 202). Comparisons are then performed to determine whether the user/job offset information stored in the user/job identification register is listed in the user/job list 100 (step 204). If the user/job information stored in the user/job identification register is not found in the user/job access list 100, then the descriptor translation process is terminated (steps 204 and 206). If the user/job offset information stored in the user/job identification register is found in the user/job access list 100, then the translation proceeds. If the page offset information 64 is not greater than the limit value, it is applied to the page table 112 identified by the page table base address (step 208). A resource base address of a resource is obtained from the page table (step 210) to which is added the offset address information 66 to obtain a real address of the resource to which access is sought (steps 210 and 212). In steps 200 and 208, if any offset is greater than the corresponding limit value, the descriptor translation process is terminated (steps 201 and 209).

Detailed Description Text (30):
By use of the apparatus of the present invention, particularly by use of descriptors which are virtual addresses of resources in the data processing system, the present invention increases the possibility that any penetration of the system can be detected and traced to the perpetrator. In other words, a system log can be kept of all accesses to the resources and all system faults since access to the resources is controlled. Further, for every access entry in the log, the user and job identification and the resource descriptor is recorded. Particularly, the present invention provides for the separation of processor privilege levels from data classifications or clearance levels by the use of user/job tables, domain tables, and page tables.